実践プログラミング
2016-06-18

{ @1: hd(nil) → nil
{ @2: hd(X|L) → X

1.
1) hd(0|1|nil)

@2 → 0

2) +1(0|1|nil)

{ @1: +1(nil) → nil
{ @2: +1(X|L) → L

@2 → (1|nil)

3) [2..5]

{ @1: [X..Y]
→ if X > Y
then { nil }
else { X|[X+1..Y]}

@1 (2>5)
→ 2|[2+1..5]

@1
→ 2|[3..5]

@1
→ 2|3|[3+1..5]

→ 2|3|[4..5]

@1
→ 2|3|4|[4+1..5]

→ 2|3|4|[5..5]

@1
→ 2|3|4|5|[5+1..5]

→ 2|3|4|5|[6..5]

@1
→
2|3|4|5|nil

レポート
- プログラム
- レポート with 説明

採点

2. gcd (2015, 31031)

{ @1 : gcd (X, 0) → X
{ @2 : gcd (X, N≥Y) → gcd (N≥Y, X rem N≥Y)

gcd (2015, 31031)

@2 → gcd (31031, 2015 rem 31031)
(2015)

@2 → gcd (2015, 31031 rem 2015)
(806)

@2 → gcd (806, 2015 rem 806)
(403)

@2 → gcd (403, 806 rem 403)
(0)

@1 → ~~gcd (403, 0)~~ 403 //

3. fact (5)

{ @1 : fact (0) → 1.
{ @2 : fact (N≥X) → N≥X × fact (P N≥X)

fact (5)

@2 → 5 × fact (P 5)

@2 → 5 × 4 × fact (P 4)

@2 → 5 × 4 × 3 × fact (P 3)

@2 → 5 × 4 × 3 × 2 × fact (P 2)

@2 → 5 × 4 × 3 × 2 × 1 × fact (P 1)

@1 → 5 × 4 × 3 × 2 × 1 × 1

4. oedc-fact(5)

{ @1: oedc-fact(0) → 1.
@2: oedc-fact(N≥X) → g(N≥X, 1).
@3: g(X,Y) = if (X>Y) then { g(X, 2×Y) × g(sd(X,Y, 2×Y) } else { X }.

※of:
oedc-fact

of(5)

@2→ g(5,1)
@3→ g(5, 2×1) × g(|5-1|, 2×1)
@3→ g(5,2) × g(4×2)
@3→ (g(5, 2×2) × g(|5-2|, 2×2)) × (g(4, 2×2) × g(|4-2|, 2×2))
@3→ (g(5,4) × g(3,4)) × (g(4,4) × g(2,4))
@3→ ((g(5, 2×4) × g(|5-4|, 2×4)) ×3) × (4×2)
@3→ ((5×1)×3) × (4×2)
→ 120 //

6. QSORT qsort(4|7|5|1|0|3|6|2|nil)

※可読性のためリストを[0,1,…]で表記している。

{ @1: qsort(nil) → nil.
@2: qsort(X|nil) → X|nil.
@3: qsort(X|Y|L) → partition(X, Y | L, nil, nil)

{ @4: partition(X, nil, LL, RL) = qsort(LL)@(X|qsort(RL)).
@5: partition(X, Y|L, LL, RL)
= if Y < X
then { partition(X, L, Y | LL, RL) }
else { partition(X, L, LL, Y | RL) }.

qsort([4,7,5,1,0,3,6,2,nil])

@3→ ~~qsort~~
partition(4, [7,5,1,0,3,6,2,nil], nil, nil)
(Y<X): false
@5→ partition(4, [5,1,…,nil], nil, [7,nil])
(Y<X): false
@5→ partition(4, [1,0,…,nil], nil, [5,7,nil])
@5 (Y<X): true
→ partition(4, [0,…nil], [1,nil], [5,7,nil])

@5 → partition(4, [3, 6, 2, nil], [0, 1, nil], [5, 7, nil])

(y<x): true — x — y — L — LL — RL

@5 → p(4, [6, 2, nil], [3, 0, 1, nil], [5, 7, nil])

x — y — L — LL — RL

@5 y<x: false

→ p...(4, [2, nil], [3, 0, 1, nil], [6, 5, 7, nil])

@5 y<x: true

→ p...(4, nil, [2, 3, 0, 1, nil], [6, ... nil])

@4 → qsort([2, 3, 0, 1, nil]) @ qsort(4 | [6, ..., nil])

[3]

@3 → partition(2, [3, 0, 1, nil], nil, nil)

@5 → p...(2, [0, 1, nil], nil, [3, nil])

@5 → p(2, [1, nil], [0, nil], [3, nil])

@5 → p(2, nil, [1, 0, nil], [3, nil])

@4 → qsort([1, 0, nil]) @ (2 | qsort([3, nil]))

[2]

@3 → part(1, [0, nil], nil, nil)

@2 → part(1, nil, [0, nil], nil)

@4 → qsort([0, nil]) @ (1 | qsort(nil))

@2 → [0, nil]

[1]

@1 → [0, nil] @ [1, nil]

@7 → 0 | (nil @ [1, nil])

@6 → 0 | ([1, nil])

[2]

@8 → [0, 1, nil] @ (2 | qsort([3, nil]))

@2 → ≤ @ (2 | [3, nil])

@8 → ≤ @ ([2, 3, nil])

@7 → 0 | ([1, nil] @ [2, 3, nil])

@7 → 0 | 1 | (nil @ [2, 3, nil])

[3]

@6 → (0 | 1 | [2, 3, nil]) @ qsort(4 | [6, ... nil])

@3 → [0, 1, 2, 3, nil] @ part(4, [5, 7, nil], nil, nil)

@5 → [≤] @ part(4, [5, 7, nil], nil, [6, nil])

x — y — L

@5 → [≤] @ part(4, [7, nil], nil, [5, 6, nil])

@6: nil @ L2 → L2

@7: (x | L) @ L2 → x | (L @ L2).

@8: Nat/NatList → NnNat List {COUNTER}

@5 → [/] @ part(4, nil, nil, [7,5,6, nil])

@4 → [/] @ (qsort(nil) @ (4 | qsort([7,5,6, nil])))

@1 → [/] @ (nil @ (4 | qsort([7, ..., nil])))

@3 → [/] @ (nil @ (4 | part(7, 5, [6, nil], nil, nil)))

@5 → [/] @ (nil @ (4 | part(7, ~~5~~ [6, nil], [5, nil], nil)))

@3 → part(7, nil, [6, 5, nil], nil)

@4 → qsort([6, 5, nil]) @ (7 | q(nil)) [4]

@3 → part(6, [5, nil], nil, nil)

@5 → ~~q(nil) @ t6~~

part(6, nil, [5, nil], nil)

@4 → qsort([5, nil]) @ (6 | qsort(nil))

@2 → [5, nil] @ (6 | q(nil))

@1 → [5, nil] @ (6 | nil)

@3 → ~~[5~~ 5 | ~~t~~ (nil @ [6, nil]) [4]

(5 | [6, nil]) @ (7 | q(nil))

@1 → (5 | [6, nil]) @ [7, nil]

@7 → 5 | [6, nil] @ (~~nil~~ [7, nil])

@7 → [/] @ (nil @ 4 | 5 | (nil @ [6, ~~nil~~, 7, nil]))

@6 → [/] @ (nil @ 4 | 5 | [6, ~~nil~~, 7, nil])

@5 → [/] @ [4, ~~6~~ 5,6, ~~5~~, 7, nil]

@7 → ~~[0~~ 0 | ([1, 2, 3, nil] @ [4 ~~6~~ 5,6, ~~5~~, 7, nil])

@7 → 0 | 1 | [2, 3, nil] @ [4, 5, 6, 7, nil]

@5 → 0 | 1 | 2 | [3, nil] @ [4, 5, 6, 7, nil]

@7 → 0 | 1 | 2 | (~~nil~~ 3, nil @ [4, /, nil])

@7 → ~~@6~~ 0 | 1 | 2 | [nil] @ 3 [4, ..., nil])

@6 → [0, 1, 2, 3, 4, 5, 6, 7, nil]

7.

ERATOSTHENES' SIEVE

@0: primesUpto (10)

@1 = primesUpto(X) → sieve ([2 .. X]).

@2 = sieve (nil) → nil.

@3 = sieve (X|L) → X | sieve (check (X, L))

@4 = check (0, L) → L.

@5 = check (NzX, nil) → nil.

@6 = check (NzX, Y|L)

→ If NzX devides Y

then { check (NzX, L) }

else { Y | check (NzX, L) }.

@7 = [X .. Y]

→ if X > Y

then { nil }

else { X | [X+1 .. Y] }.

@0: primeUpto (10)

@7 → sieve ([2..10])

@7 → sieve (2 | [3 .. 10])

@7 → sieve (2|3|[4..10])

@7 → 2|3|4|[5..10]

@7 → 2|3|4|5|6|[7..10]

@7 → 2|3|4|5|6|7|[8..10]

@7 → 2|3|4|5|6|7|8|[9..10]

@7 → 2| // |9| [10..10]

@7 → 2| // |9|10| [11..10]

@7 → sieve (2| // |9|10| nil)

@7 → 2 | sieve (check (2, [3, 4, 5, 6, 7, 8, 9, 10, nil]))

(NzX, Y)

@6 → 2 | sieve (3 | check (2, [4, 5, .., nil]))

(NzX, Y)

@6 → 2 | sieve (3 | check (2, [5, 6, .., nil]))

@6 → 2 | sieve (3 | 5 | check (2, [6, 7, .., nil]))

@6 → 2 // (5 | check (2, [7, 8, .., nil]))

@6→ 2| sieve(3|5|7|check(2, [8, 9, 10, nil]))

@6→ 2| // check(2, [9, 10, nil])

@6→ 2| // |9| check(2, [10, nil])

@6→ 2| // check(2, nil)

@5→ 2| sieve(3|5|7| nil)

@3→ 2|3| sieve(check(3, [5, 7, nil]))

@6→ 2|3| sieve(5|check(3, [7, nil]))

@6→ 2|3| sieve(5|7| check(3, nil))

@5→ 2|3| sieve(5|7| nil)

@3→ 2|3|5| sieve(check(5, [7, nil]))

@6→ 2| // | sieve(7| check(5, nil))

@5→ 2| // | sieve(7|nil)

@3→ 2| // |7|sieve(check(7, nil))

@5→ 2|3|5|7| sieve(nil)

@2→ 2|3|5|7| nil

//